Université de Poitiers, Année universitaire 2024–2025
U.F.R. Sciences Fondamentales Appliquées
Master 1 TDSI - parcours Objets Connectés

Montre Connectée

Projet Module Systèmes Embarqués Communicants

DJENADOU Rayane & DJESSOU Koundeme Nobel & MORET Maxime

Module enseigné par BOEGLEN Hervé & RICHARD Noel

Table des matières

Table des figures

Introduction

Le présent rapport détaille le développement d'une montre connectée basée sur le microcontrôleur nrf5340, s'inspirant du projet open-source ZSWatch. Ce projet s'inscrit dans le cadre d'une application pratique des systèmes embarqués combinant hardware et software pour créer un dispositif portable connecté.

La ZSWatch, développée initialement par Jakob Krantz et Daniel Kampert, représente une approche remarquable de conception complète d'une montre intelligente, tant au niveau matériel que logiciel. Son architecture repose sur le système d'exploitation temps réel Zephyr, d'où son nom "Zephyr Smartwatch". Notre projet reprend les principes fondamentaux de cette réalisation tout en adaptant certains aspects pour répondre aux contraintes spécifiques de notre cahier des charges.

L'objectif principal est de concevoir et implémenter une montre connectée fonctionnelle intégrant diverses technologies : capteurs environnementaux et de mouvement, communication Bluetooth Low Energy, interface graphique tactile et gestion du temps précise via un module RTC dédié. Cette approche modulaire permet d'aborder les différents aspects des systèmes embarqués modernes, depuis la gestion des périphériques matériels jusqu'à l'interaction utilisateur.

Ce rapport présente dans un premier temps les caractéristiques techniques de la plateforme nrf5340 utilisée, puis détaille la gestion des différents capteurs intégrés. Les phases de développement et de validation unitaire de chaque composant sont ensuite abordées, suivies par l'intégration et la validation de l'ensemble du système. Une attention particulière est portée à l'architecture logicielle, notamment à la machine d'états permettant de gérer efficacement les différentes fonctionnalités de la montre.

Cette réalisation, bien que simplifiée par rapport au projet ZSWatch original, vise à mettre en œuvre les concepts fondamentaux des systèmes embarqués connectés tout en offrant une expérience utilisateur proche de celle des montres intelligentes commerciales.

1 Cahier des charges

Le cahier des charges de notre projet de montre connectée s'inspire directement du projet ZSWatch tout en l'adaptant à nos contraintes de temps et de ressources. Cette section détaille les spécifications fonctionnelles et techniques que notre implémentation doit satisfaire.

1.1 Table des Entrées/Sorties

Notre montre connectée doit intégrer plusieurs interfaces d'entrée et de sortie permettant l'interaction avec l'utilisateur et l'environnement. Ces éléments sont énumérés ci-dessous :

- **Entrées :**
  - Écran tactile capacitif comme interface principale d'entrée utilisateur
  - Capteurs LSM6DSO (accéléromètre et gyroscope) pour la détection de mouvements
  - Capteur LIS2MDL (magnétomètre) pour l'orientation
  - Capteurs environnementaux pour la mesure de température, humidité et pression
  - Module RTC RV-8263-C8 pour le maintien de l'heure et de la date
- **Sorties :**
  - Écran graphique 320x240 pixels pour l'affichage d'informations
  - Module Bluetooth Low Energy pour la communication avec un smartphone
  - Indicateurs visuels pour les notifications et alertes

1.2 Fonctionnalités requises

Le projet doit implémenter les fonctionnalités suivantes, en accord avec le document fourni :

1. Utilisation de l'OS temps réel Zephyr comme base logicielle
2. Implémentation d'une interface graphique via le framework LVGL sur l'écran tactile 320x240
3. Intégration et exploitation des capteurs d'environnement et de mouvement
4. Communication Bluetooth Low Energy pour l'échange de données avec un smartphone
5. Gestion du temps via le module RTC externe (RV-8263-C8)

1.3 Contraintes de développement

Notre implémentation doit respecter plusieurs contraintes :

- Utilisation exclusive du microcontrôleur nrf5340 comme processeur principal
- Développement basé sur le système d'exploitation Zephyr
- Utilisation du shield ST IKS01A3 pour les capteurs
- Exploitation de l'écran Adafruit 2.8" TFT Touch Shield v2
- Conception graphique réalisée avec le logiciel Squareline Studio
- Respect des contraintes énergétiques inhérentes aux dispositifs portables
- Conformité aux exigences de performance temps réel pour une interaction fluide

1.4 Livrables attendus

À l'issue du projet, les éléments suivants devront être fournis :
— Code source complet et documenté de l'application
— Documentation technique détaillant l'architecture matérielle et logicielle
— Manuel d'utilisation de la montre connectée
— Démonstration fonctionnelle des principales fonctionnalités
— Rapport d'analyse détaillant la machine d'états et l'utilisation de Zephyr

Cette adaptation du projet ZSWatch constitue un défi technique significatif, nécessitant l'intégration de multiples technologies dans un facteur de forme réduit, tout en assurant une expérience utilisateur fluide et intuitive.

2 Choix des capteurs et périphériques

La sélection des capteurs pour notre montre connectée est principalement guidée par les fonctionnalités requises définies dans le cahier des charges (section **??**) et par le matériel spécifique mis à notre disposition ou spécifié pour le projet. Cette section détaille les composants retenus pour la mesure du mouvement, de l'orientation et du temps.

2.1 Capteurs de Mouvement et d'Orientation (IMU)

Pour la détection des mouvements et le calcul de l'orientation, nous utilisons les capteurs présents sur le shield d'extension `STMicroelectronics IKS01A3`, conformément aux contraintes de développement. Ce shield intègre une unité de mesure inertielle (IMU) complète :

— `LSM6DSO` : Il s'agit d'un composant 6 axes combinant un accéléromètre 3 axes et un gyroscope 3 axes.
  — L'accéléromètre mesure l'accélération linéaire (statique et dynamique), permettant de détecter l'orientation par rapport à la gravité, les chocs, les vibrations ou les pas (podomètre).
  — Le gyroscope mesure la vitesse angulaire, essentielle pour suivre les rotations de la montre dans l'espace.
— `LIS2MDL` : C'est un magnétomètre 3 axes qui mesure le champ magnétique terrestre. Combiné aux données de l'accéléromètre et du gyroscope (via des algorithmes de fusion de données, bien que leur implémentation complexe ne soit pas un objectif premier), il permet d'obtenir une orientation absolue (cap).

Ces capteurs communiquent typiquement via le bus `I2C` avec le microcontrôleur `nrf5340`. Le système `Zephyr` fournit des pilotes (drivers) et une API de capteurs (`sensor_api`) qui facilitent leur intégration et la lecture des données.

2.2 Capteurs Environnementaux

Le cahier des charges mentionne la possibilité d'intégrer des capteurs environnementaux pour mesurer la température, l'humidité et la pression atmosphérique. Il est important de noter que le shield `IKS01A3` utilisé **intègre** le capteur HTS221 qui nous permet de mesurer la température et l'humidité.

2.3 Module Horloge Temps Réel (RTC)

La gestion précise et persistante du temps est cruciale pour une montre. Le microcontrôleur `nrf5340`, bien qu'il dispose de timers internes, ne possède pas d'horloge temps réel (RTC) native capable de fonctionner de manière autonome sur une pile bouton ou une batterie de secours avec une très faible consommation. Pour pallier ce manque et assurer que l'heure et la date soient maintenues même lorsque le processeur principal est en mode veille profonde ou brièvement hors tension, le projet impose l'utilisation d'un module RTC externe dédié :

— `RV-8263-C8` de Micro Crystal : Ce composant est un module RTC de haute précision et très basse consommation. Il gère le calendrier complet (secondes, minutes, heures, jour, date, mois, année) et peut fournir des alarmes ou des interruptions périodiques.

La communication avec ce module se fait via le bus `I2C`. Son intégration nécessite soit une carte fille (breakout board) spécifique, soit un routage personnalisé si un PCB dédié était réalisé. L'utilisation de ce module est une fonctionnalité requise par le projet (section **??**).

2.4 Ecran Tactile

Bien qu'il s'agisse d'une interface d'entrée utilisateur principale plutôt que d'un capteur environnemental, le capteur tactile de l'écran est un composant essentiel.

— **Contrôleur tactile capacitif** : L'écran `Adafruit 2.8" TFT Touch Shield v2` est équipé d'une surface tactile resistive. Le contrôleur spécifique gérant cette surface (par exemple, un `STMPE610` ou un `FT6x06`) détecte la position des doigts de l'utilisateur.

Ce contrôleur communique généralement via `I2C` avec le `nrf5340`. L'intégration de ce capteur est indispensable pour l'interaction avec l'interface graphique `LVGL`. `Zephyr` et `LVGL` disposent de mécanismes pour gérer les événements d'entrée provenant de ce type de périphérique.

3 Analyse Préliminaire

Avant d'entamer le développement détaillé de chaque module, une phase d'analyse préliminaire est essentielle pour définir l'architecture globale du système, les flots de données et d'événements, ainsi que la logique comportementale principale. Cette démarche permet d'anticiper les interactions complexes, de valider certains choix technologiques et d'établir une base de conception solide.

3.1 Contexte Matériel, Logiciel et Fabrication

Cette section répond à quelques questions fondamentales concernant le périmètre et les outils du projet.

3.1.1 Fonctionnalités et Architecture Globale (Question 1.1)

Un schéma bloc fonctionnel est présenté en Figure 1.

Catégorie	Fonctionnalités
Communication	Bluetooth Low Energy (BLE) pour la communication avec un smartphone (ex : via Gadgetbridge)
Capteurs	— BME688 : capteur environnemental (température, humidité, gaz, pression) — BMP581 : capteur de pression haute précision — BMI270 : IMU (accéléromètre + gyroscope) — LID2MDLTR : magnétomètre — MAX30101 : oxymètre de pouls / fréquence cardiaque
Interface utilisateur	— Écran couleur 240x240 avec écran tactile capacitif — Haptic feedback via le driver DRV2603 — UI basée sur LVGL
Gestion de l'alimentation	— Mise en veille automatique après inactivité — Réveil par mouvement (IMU) ou interaction — Coupure d'alimentation de l'écran — Gestion fine des tâches périodiques (Fast/Medium/Slow)
Architecture logicielle	— Système modulaire avec Zephyr RTOS — Communication interne via ZBUS — Code organisé en "Apps" indépendantes — Exécution sur nRF5340 (SoC Nordic)

TABLE 1 – Fonctionnalités principales de la ZSWatch

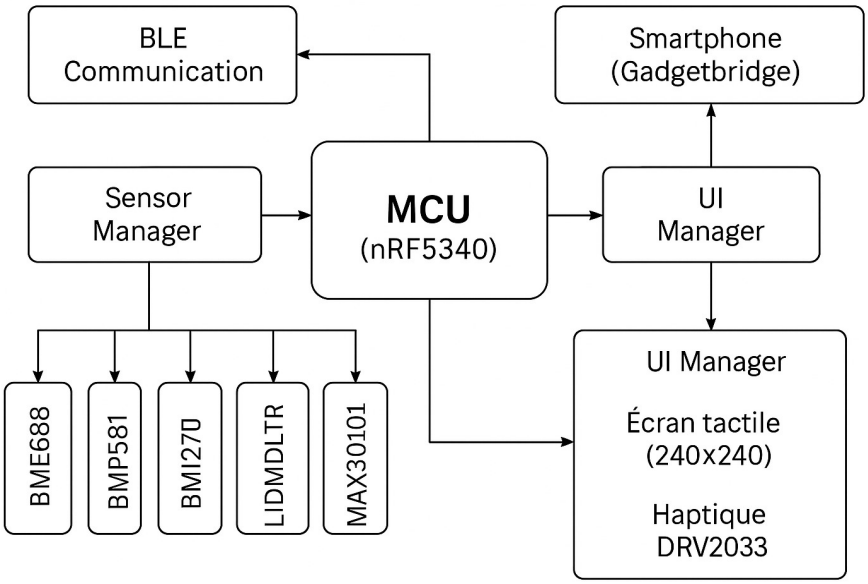


FIGURE 1 – Schéma bloc fonctionnel de la montre connectée.

3.1.2 Outils de Conception Électronique (Question 1.2)

Le logiciel utilisé pour la saisie du schéma électrique et le routage de la carte électronique (PCB) est **KiCad**.

3.1.3 Contraintes de Fabrication du PCB (Question 1.3)

La fabrication du PCB de la montre présente plusieurs contraintes typiques des systèmes embarqués compacts :

- **Nombre de couches :** Probablement 4 couches ou plus pour gérer la densité des composants et les signaux (alimentation, masse, signaux rapides, RF).
- **Taille des composants :** Utilisation de composants CMS (SMD) de petite taille (ex : 0402, 0201) et potentiellement des boîtiers complexes à pas fin (QFN, BGA pour le nRF5340 ou d'autres puces).
- **Largeur des pistes et isolements :** Contraintes sur les largeurs minimales des pistes et les distances d'isolement en raison de la miniaturisation.
- **Impédance contrôlée :** Nécessaire pour l'antenne BLE RF et potentiellement pour d'autres signaux haute fréquence (ex : bus SPI de l'écran).
- **Via :** Utilisation probable de vias de petite taille, potentiellement des vias borgnes ou enterrés sur des PCB plus coûteux.

Concernant la réalisabilité par un amateur, elle est **probablement difficile** sans équipement spécialisé. L'assemblage manuel de composants CMS fins (QFN, BGA) nécessite généralement une bonne expérience, une loupe binoculaire ou un microscope, et idéalement un four à refusion ou une station à air chaud. La fabrication du PCB lui-même (gravure, perçage, métallisation) requiert des procédés industriels pour atteindre la précision nécessaire, bien que des services de fabrication de prototypes soient accessibles.

3.1.4 Environnement de Développement Logiciel (Question 1.4)

L'environnement logiciel utilisé pour le développement applicatif est basé sur :

- Le **nRF Connect SDK** de Nordic Semiconductor, qui intègre :
- Le système d'exploitation temps réel (RTOS) **Zephyr Project**.
- Les drivers matériels, les bibliothèques (gestion BLE, capteurs, etc.) et les middlewares.
- Le langage de programmation principal est le **C**.
- L'environnement de développement intégré (IDE) utilisé est **Visual Studio Code** .

3.1.5 Méthode de Programmation de la Cible (Question 1.5)

Le transfert du programme compilé (firmware) vers le microcontrôleur nRF5340 de la montre se fait via une interface de débogage standard :

- **Interface Matérielle :** SWD (Serial Wire Debug), une variante de JTAG optimisée pour les cœurs ARM Cortex-M.
- **Outil Matériel :** Une sonde de débogage/programmation (ex : J-Link de Segger, sonde intégrée sur les kits de développement Nordic DK).
- **Connexion :** La sonde est connectée au PC via USB et à la carte cible via un connecteur SWD dédié (souvent un connecteur à broches de petit pas).
- **Outil Logiciel :** La commande `west flash` (outil en ligne de commande de Zephyr/nRF Connect SDK) ou le bouton de programmation ("Flash") directement

depuis l'IDE (VS Code, Segger Embedded Studio) qui utilise en arrière-plan les utilitaires de la sonde (ex : nrfjprog, J-Flash Lite).

3.2 Organisation du système

Le système de la montre connectée peut être décomposé logiquement en plusieurs sous-systèmes principaux qui interagissent entre eux pour réaliser les fonctionnalités requises :

- **Système d'Acquisition :** Responsable de l'interface avec les capteurs (IMU `LSM6DSO/LIS2MDL`, RTC `RV-8263-C8`) et le module tactile. Il inclut la configuration des périphériques via `I2C/SPI` et la récupération des données brutes ou des événements (touch).
- **Système de Traitement :** Cœur logique de l'application. Il exécute le `Zephyr RTOS`, gère les tâches (threads), traite les données des capteurs (filtrage simple, formatage), implémente la machine d'états principale, gère la communication `BLE` et l'heure système.
- **Système d'Interface Utilisateur (UI) :** Gère l'affichage graphique sur l'écran LCD via `LVGL`. Il reçoit les données formatées du système de traitement et les événements d'entrée (tactile) pour mettre à jour l'interface et interagir avec l'utilisateur. Il inclut également la gestion des écrans conçus avec `Squareline Studio`.
- **Système de Communication :** Gère spécifiquement la couche `Bluetooth Low Energy`, incluant l'initialisation du stack, l'advertising, la gestion des connexions, et la définition/gestion des services et caractéristiques `GATT`.
- **Système de Gestion de l'Énergie :** (Moins explicite mais important) Comprend la configuration des modes de faible consommation du `nrf5340` et des périphériques, gérés par `Zephyr` et l'application.

Ces sous-systèmes ne correspondent pas nécessairement à des modules logiciels distincts mais représentent des domaines fonctionnels clés.

3.3 Ébauche de l'Interface Utilisateur

Une première conception de l'interface utilisateur graphique a été réalisée afin de visualiser l'agencement général et les informations clés à présenter sur les différents écrans. La Figure 2 montre ces écrans initiaux dessinés à la main.

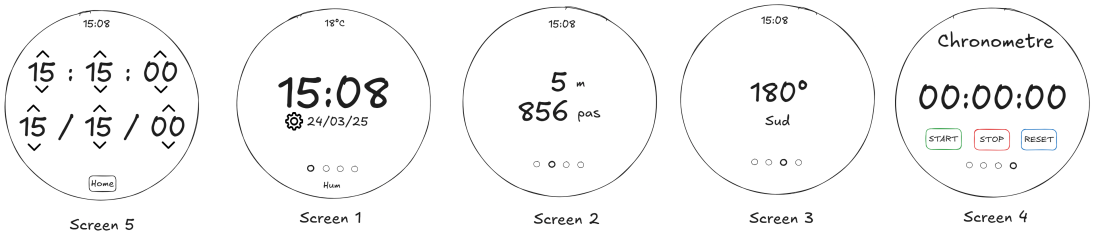


FIGURE 2 – Ébauche des écrans principaux de l'interface utilisateur (Dessin).

Les écrans principaux envisagés dans cette ébauche (Figure 2) sont :

- **Screen 1 (Accueil) :** Affiche l'heure actuelle (`15:08`), la date (`24/03/25`), une icône de réglage, et potentiellement la température ambiante (`18°C`) dans une barre d'état supérieure. Des points de navigation indiquent la position dans la séquence d'écrans.

— **Screen 2 (Activité/Pas) :** Présente le nombre de pas effectués (`856 pas`) et une estimation de la distance (`5 m`). L'heure est rappelée en haut.
— **Screen 3 (Orientation/Boussole) :** Indique l'orientation actuelle sous forme d'angle (`180°`) et de point cardinal (`Sud`). L'heure est rappelée en haut.
— **Screen 4 (Chronomètre) :** Fournit une interface de chronomètre avec affichage du temps écoulé (`00:00:00`) et des boutons `START`, `STOP`, `RESET`.
— **Screen 5 (Réglage Heure/Date) :** Permet à l'utilisateur de modifier l'heure et la date à l'aide de boutons ou zones tactiles (symbolisés par des flèches). Un bouton "Home" permet de revenir à l'écran d'accueil.

Cette ébauche sert de guide pour l'implémentation détaillée avec LVGL et Squareline Studio, en définissant les informations et les interactions de base pour chaque écran.

3.4 Assignation des Broches (Pinout)

L'interfaçage entre le microcontrôleur nRF5340 et les différents périphériques (capteurs, écran, tactile) est réalisé via les broches GPIO configurées pour des protocoles spécifiques (I2C, SPI). Dans notre cas, utilisant une carte de développement **[Nom de votre carte nRF5340, ex : nRF5340 DK]** avec un shield Nucleo **[Nom/Référence du shield, ex : X-NUCLEO-IKS01A3]** connecté via les headers Arduino R3, le mapping entre les broches du shield et celles du nRF5340 est primordial. Le tableau 2 résume les assignations principales utilisées, basées sur le mapping standard des headers Arduino du nRF5340 DK et les connexions typiques du shield. Ces assignations sont confirmées et définies précisément dans le fichier d'overlay DeviceTree ('.overlay') du projet Zephyr.

TABLE 2 – Assignation des broches principales (basée sur nRF5340 DK).

Périphérique	Interface	Broches nRF5340
HTS221 (Temp/Hum)	I2C	P1.03(SCL), P1.02(SDA)
LSM6DSO (IMU)	I2C	P1.03(SCL), P1.02(SDA)
LIS2MDL (Magnéto)	I2C	P1.03(SCL), P1.02(SDA)
RV-8263-C8 (RTC)	I2C	P1.03(SCL), P1.02(SDA)
Écran TFT	SPI	SCK : D13, MOSI : D11 CS :D10 , DC : D9
Touchscreen	I2C	P1.03(SCL), P1.02(SDA)

3.5 Diagramme des bords du modèle

Ce diagramme illustre les interfaces matérielles et logicielles directes du microcontrôleur `nrf5340` (le cœur du modèle) avec les composants externes (périphériques) et les services logiciels de haut niveau. Il permet de visualiser les points d'entrée/sortie du système principal.

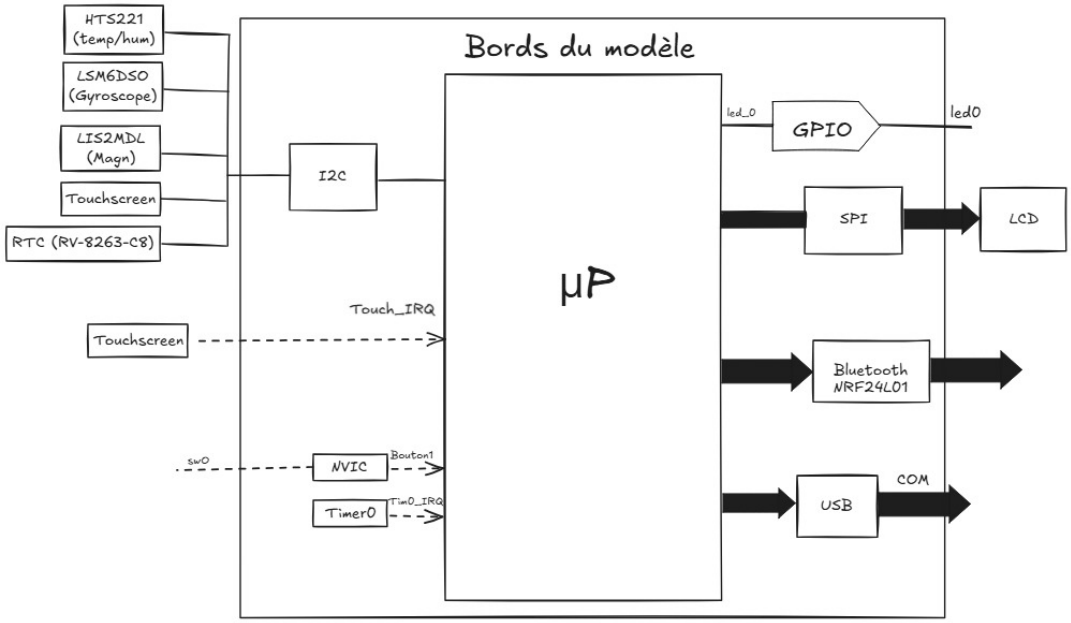


FIGURE 3 – Diagramme des bords du modèle montrant les interactions du nRF5340.

3.6 Diagramme de flot de données (DFD)

Le diagramme de flot de données met en évidence le cheminement des informations à travers les différents processus (fonctions logicielles, tâches) et les stockages de données (variables, buffers) du système. Il aide à comprendre comment les données brutes des capteurs sont transformées en informations utiles pour l'utilisateur ou la communication BLE.

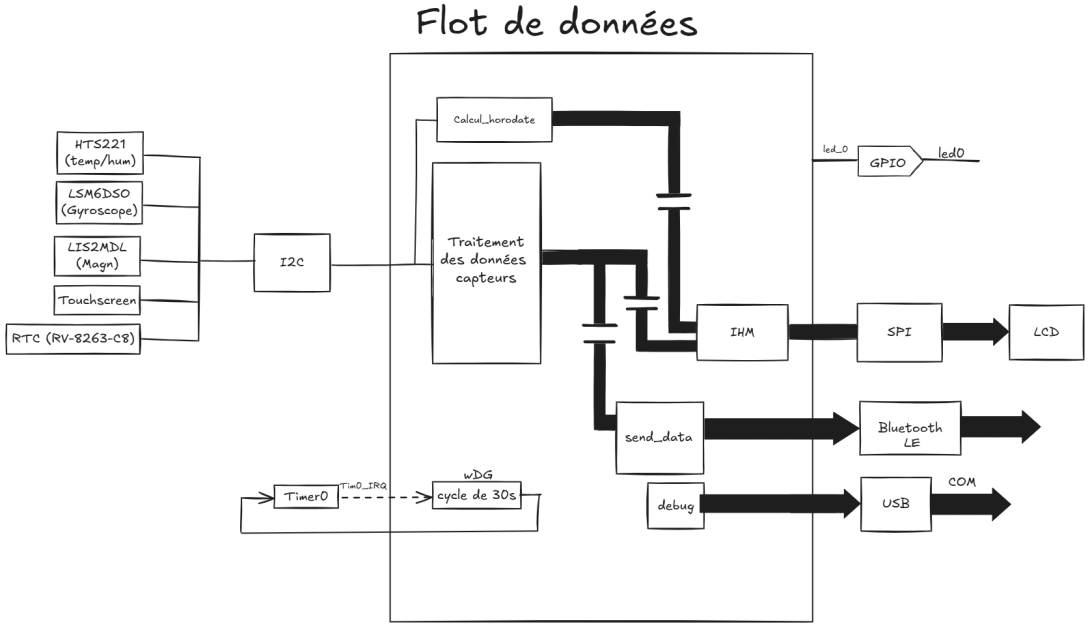


FIGURE 4 – Diagramme de flot de données du système de la montre.

3.7 Diagramme de flot d'événements

Ce diagramme se concentre sur les événements qui déclenchent des actions dans le système. Les événements peuvent provenir de sources externes (appui tactile, réception BLE, interruption capteur, alarme RTC) ou internes (expiration d'un timer Zephyr, signal entre threads). Il montre la séquence typique des interactions déclenchées par ces événements.

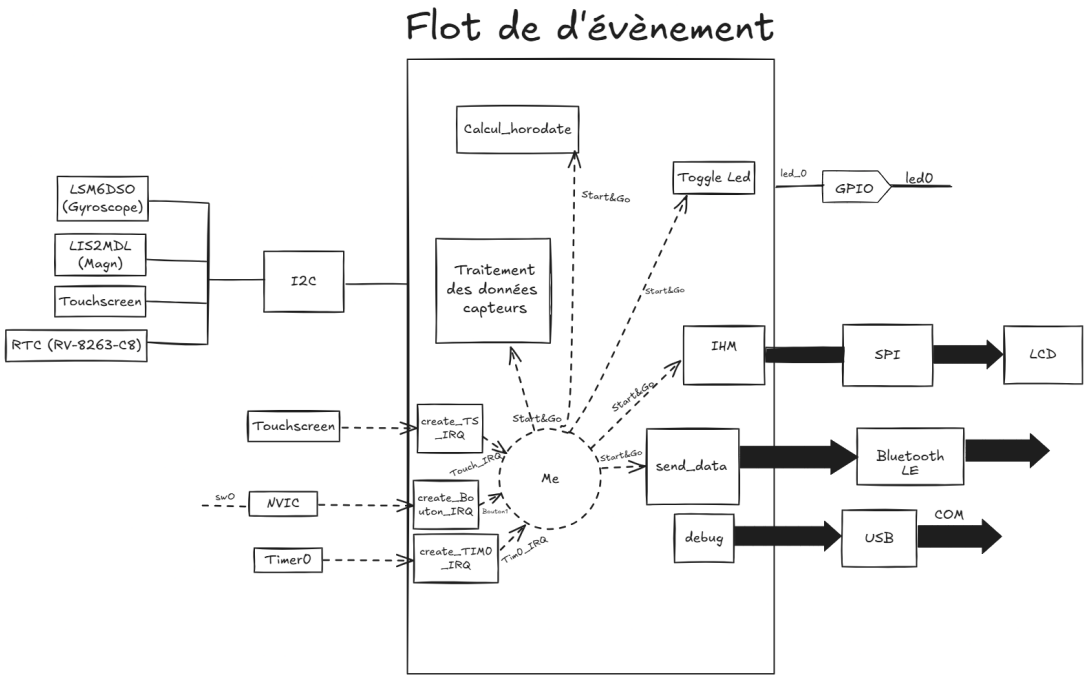


FIGURE 5 – Diagramme de flot de d'évenements

3.8 Machine d'états

La logique comportementale globale de la montre peut être modélisée par une machine d'états finis . Elle décrit les différents modes de fonctionnement de la montre (ex : Affichage Heure, Menu Capteurs, Connexion BLE, Configuration, Veille) et les transitions entre ces états, déclenchées par les événements identifiés précédemment. Ceci est crucial pour structurer l'application principale et gérer son comportement de manière cohérente.

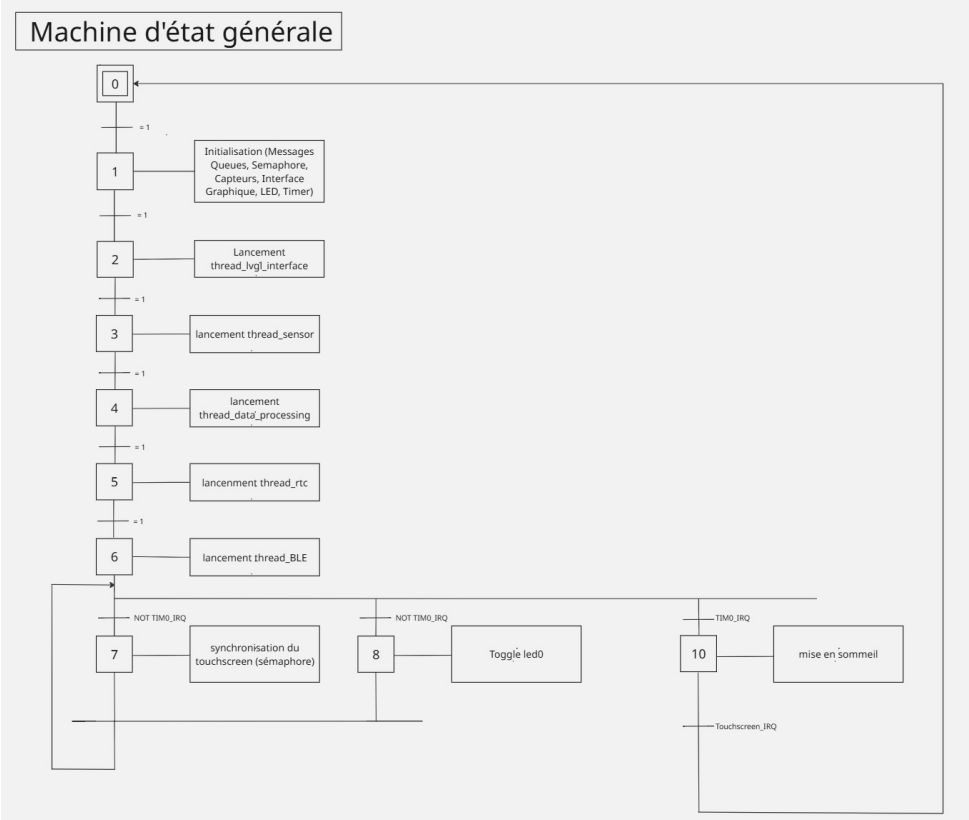


FIGURE 6 – Machine d'états principale

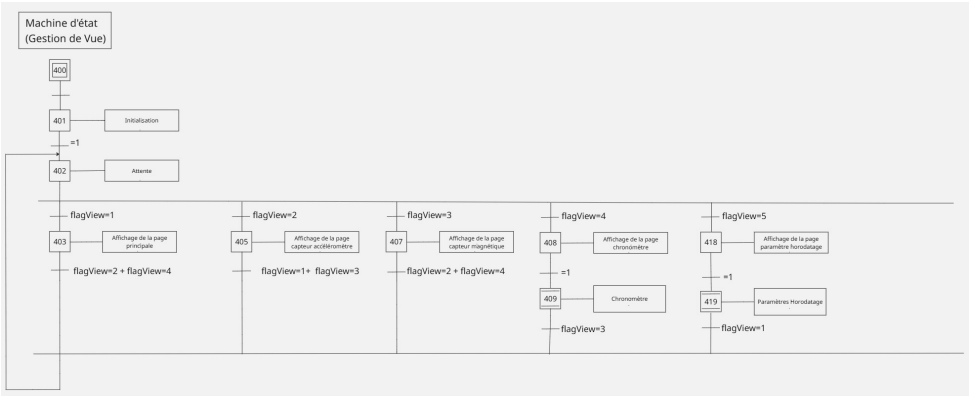


FIGURE 7 – Gestions des vues

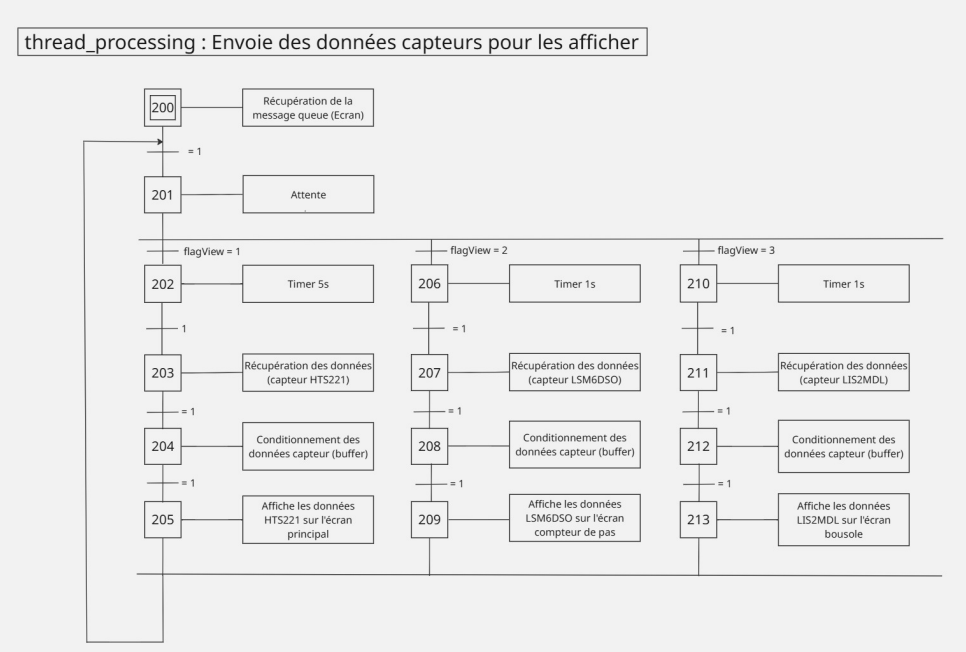


FIGURE 8 – Affichage des données capteurs

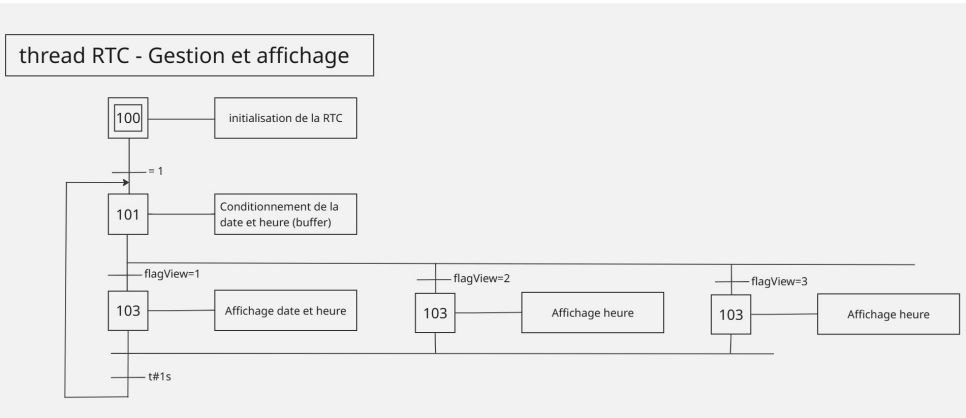


FIGURE 9 – Affichage de la date et de l'heure

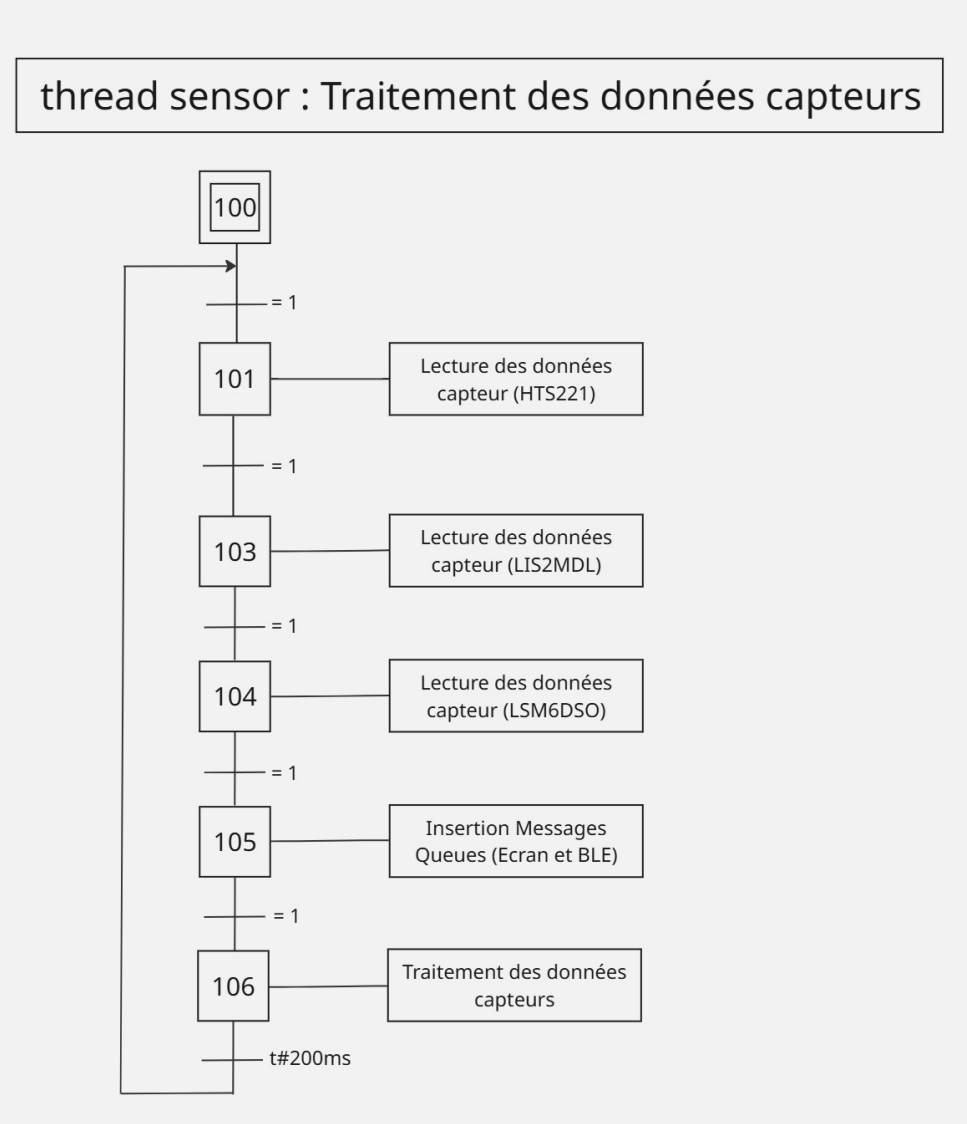


FIGURE 10 – Gestion des capteurs

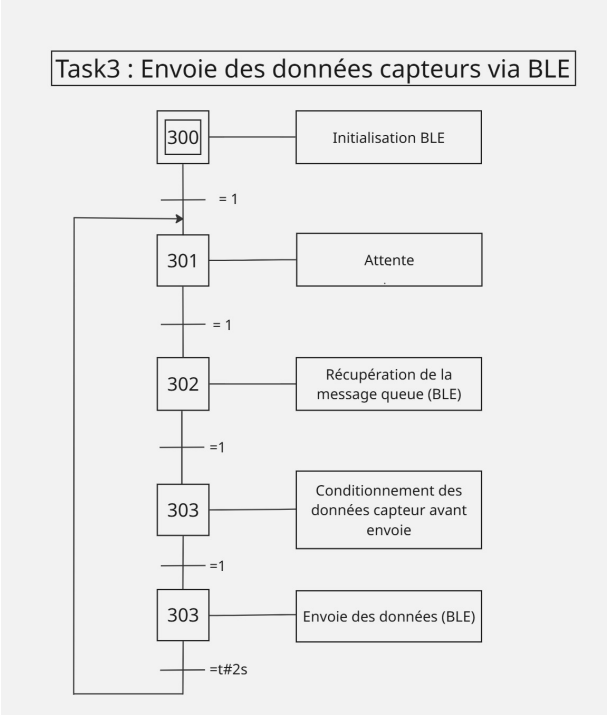


FIGURE 11 – Envoie par BLE

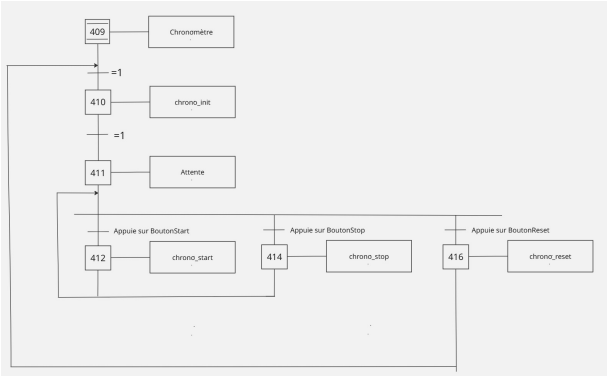


FIGURE 12 – Chronomètre

3.9 Gestion des Tâches et Priorités

Le système `Zephyr RTOS` est basé sur des tâches (threads) préemptives. Il est important de définir les threads principaux de l'application et d'assigner leurs priorités de manière judicieuse pour garantir la réactivité des fonctions critiques (ex : UI, communication BLE) tout en permettant l'exécution des tâches de fond (ex : lecture périodique des capteurs). Les interruptions matérielles (timers, GPIO, périphériques) ont également des niveaux de priorité qui interagissent avec les priorités des threads.

TABLE 3 – Tableau préliminaire des threads applicatifs et priorités.

Thread / Tâche Principale	Priorité Zephyr	Fonction
Thread UI (LVGL Handler)	0	Gérer de l'écran
Thread BLE	3	Gérer la communication BLE (connexion, data)
Thread Lecture Capteurs	1	Lire périodiquement les capteurs I2C
Thread Traitement des capteurs	2	Traitement de fond, logique applicative
Thread RTC	2	Gestion Heure, Date et Chronomètre

Cette analyse préliminaire, notamment la définition de la machine d'états et la structuration en threads, constitue la base de la partie "analyse" requise pour ce projet, complémentaire à l'implémentation détaillée qui suivra.

4 Développement et validation unitaire

Ce chapitre détaille le processus de développement et de test individuel des principaux composants logiciels et matériels de la montre connectée. L'approche modulaire adoptée vise à garantir le bon fonctionnement de chaque bloc avant l'intégration complète du système, conformément à la stratégie de test décrite ci-dessous.

4.1 Stratégie de test unitaire

Avant l'intégration système, chaque fonctionnalité majeure (gestion des capteurs, interface graphique, communication BLE, gestion du temps) a fait l'objet d'un développement et d'une validation dédiés. La stratégie générale a consisté à :

1. **Configuration :** Mettre en place la configuration matérielle (via DeviceTree overlays) et logicielle (via Kconfig) nécessaire pour le composant spécifique dans l'environnement `Zephyr`.
2. **Implémentation :** Écrire le code applicatif utilisant les API `Zephyr` (drivers, services BLE, LVGL) pour interagir avec le composant ou implémenter la fonctionnalité. Souvent réalisé via des exemples minimalistes ou des modules dédiés.
3. **Validation :** Vérifier le bon fonctionnement du composant ou de la fonctionnalité par des tests ciblés, en documentant les résultats par des captures d'écran ou des observations directes :
   — Lecture de logs via la console (`RTT` ou `UART`).
   — Observation du comportement matériel (ex : affichage écran, LED).
   — Utilisation d'outils externes (ex : application smartphone `nRF Connect` pour BLE, débogueur `GDB`).
   — Comparaison des sorties avec les valeurs attendues (ex : données capteurs, heure RTC).

Cette approche a permis d'identifier et de corriger les problèmes spécifiques à chaque module de manière précoce.

4.2 Mise en place de l'Environnement Zephyr

Le développement s'appuie entièrement sur `Zephyr RTOS`. Les étapes initiales ont consisté à configurer l'environnement de développement (SDK, `west`) et à créer une appli-

cation de base pour la carte `nRF5340 DK`. Les configurations clés (`prj.conf`, `app.overlay`) ont été adaptées itérativement pour chaque module développé. L'utilisation de `VS Code` avec les extensions `Zephyr Tools` a facilité la compilation, le flashage et le débogage via le J-Link intégré.

4.3 Capteurs I2C (HTS221, LSM6DSO, LIS2MDL)

L'intégration des capteurs connectés via le bus I2C a été réalisée pour acquérir les données environnementales et de mouvement. Les capteurs principaux utilisés sont le `HTS221` (température et humidité), le `LSM6DSO` (accéléromètre 6 axes et gyroscope) et le `LIS2MDL` (magnétomètre 3 axes), typiquement présents sur des shields comme le `X-NUCLEO-IKS01A3`.

4.3.1 Configuration

La configuration logicielle a impliqué :
— L'activation du contrôleur `I2C`
— L'activation des drivers Zephyr spécifiques aux capteurs.
— La définition des instances de ces capteurs dans le fichier d'overlay DeviceTree (`app.overlay`), en spécifiant leur adresse I2C et en les liant au bon contrôleur `I2C` de la carte `nRF5340 DK` (connecté via les headers Arduino).

4.3.2 Comment configurer la boussole et calculer le pas

L'objectif est de déterminer l'**orientation angulaire** (ou cap) en degrés à partir des composantes brutes X et Y du champ magnétique terrestre, mesurées par un magnétomètre (par exemple le LIS2MDL). Cette orientation angulaire est ensuite convertie en une **direction cardinale** (Nord, Sud, Est, Ouest, etc.), compréhensible par l'utilisateur.

Ce traitement est essentiel pour afficher un indicateur de direction (boussole) dans une application embarquée, notamment dans le cadre d'une montre connectée ou d'un système de navigation portatif.

Avant de procéder au calcul de l'angle, les valeurs brutes du magnétomètre sont corrigées par un **offset** appliqué à chaque axe. Cette calibration manuelle consiste à identifier les valeurs minimales et maximales sur chaque axe, observées lors d'une rotation complète du dispositif. Les offsets sont ensuite calculés comme la moyenne de ces extrêmes.

Cette étape permet de corriger les **biais statiques** introduits par :
— des erreurs internes au capteur,
— des perturbations environnementales permanentes (ferromagnétisme, interférences),
— ou des distorsions magnétiques (effets *hard iron / soft iron*).

Le centrage des données autour de zéro améliore significativement la précision de l'orientation estimée.

L'angle d'orientation est calculé à l'aide d'une fonction trigonométrique qui retourne l'angle entre l'axe horizontal et la direction du vecteur magnétique corrigé. Une convention est utilisée pour que l'axe $-X$ corresponde à l'Est, ce qui permet de faire correspondre 0° au Nord.

L'angle obtenu est converti en **degrés** pour être interprété plus facilement par l'utilisateur. Ce calcul retourne une valeur dans l'intervalle $[-180°, +180°]$, qu'il faut ensuite **normaliser**.

L'angle est normalisé pour appartenir à l'intervalle $[0°, 360°[$. Une **correction angulaire** est ensuite appliquée (par exemple un décalage de 100°) afin de :

— compenser une orientation physique différente du capteur dans son boîtier (montage mécanique),
— ajuster l'angle en fonction de la **déclinaison magnétique locale**, c'est-à-dire l'écart entre le Nord géographique et le Nord magnétique.

L'angle en degrés est ensuite utilisé pour déterminer une **direction cardinale simplifiée** (Nord, Nord-Est, Est, etc.).

La méthode divise le cercle complet en **8 secteurs de 45°**, centrés autour des directions principales. Un seuil de $\pm 22.5°$ autour de chaque cap permet de classifier l'orientation mesurée dans l'un de ces secteurs.

Ce traitement rend l'**interface de boussole plus lisible** et adaptée à une présentation utilisateur simple.

Fonction Podomètre

Le module podomètre a pour but de **détecter un pas** à partir des mesures d'accélération dans les trois axes (a_x, a_y, a_z), puis de **calculer une distance approximative** parcourue.

Cette fonctionnalité est essentielle dans les dispositifs de suivi de l'activité physique, comme les montres connectées ou les bracelets de fitness.

La détection repose sur le calcul de la **norme vectorielle de l'accélération**. Cela fournit une valeur scalaire reflétant l'intensité du mouvement global, indépendamment de l'orientation du capteur.

Lorsqu'un utilisateur marche, cette norme présente des **pics caractéristiques** à chaque impact du pied au sol. L'algorithme s'appuie sur cette signature pour détecter les pas.

- Logique de détection d'un pas Un pas est détecté lorsqu'une **montée brutale de la norme** dépasse un seuil prédéfini (par exemple 12.5 m/s^2), à condition que :

— la norme précédente était inférieure à ce seuil,
— un **délai minimum** s'est écoulé depuis la dernière détection (par exemple 600 ms), afin d'éviter les doubles comptages liés aux vibrations ou à des mouvements rapides non liés à la marche.

Après la détection, la recherche de pas est désactivée jusqu'à ce que la norme repasse en dessous d'un **seuil de retour au calme** (par exemple 10.2 m/s^2). Cette **hystérésis** entre les deux seuils augmente la robustesse de la détection.

- Estimation de la distance : À chaque pas détecté, une **longueur de pas moyenne** est ajoutée à la distance totale parcourue. Cette longueur, fixée par défaut à 0.7 m, peut être personnalisée selon l'utilisateur (taille, mode de déplacement, etc.).

Le cumul du nombre de pas et de la distance parcourue peut ensuite être affiché à l'écran ou transmis via Bluetooth.

4.3.3 Structure `Sensor_Data`

Cette structure a pour but de regrouper en un seul endroit l'ensemble des données brutes (converties en `double`) provenant des différents capteurs I^2C (HTS221, LSM6DSO, LIS2MDL). Elle est utilisée comme membre de la structure `sensor_msg_data` pour le transfert de données entre threads via une file de messages.

Définition C : Sensor_Data

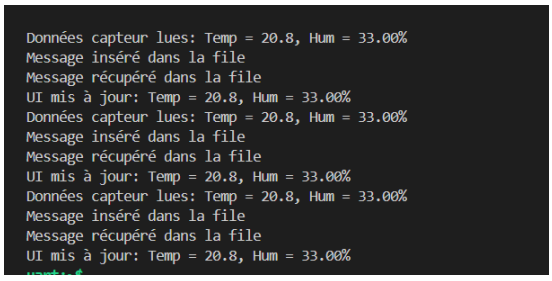
```c
/* Structure des données capteurs aggrégées */
typedef struct{
```

```
    double temperature; // Température (HTS221)
    double humidity;    // Humidité (HTS221)
    double Mx, My, Mz;  // Magnétomètre X, Y, Z (LIS2MDL)
    double ax, ay, az;  // Accéléromètre X, Y, Z (LSM6DSO)
    double gx, gy, gz;  // Gyroscope X, Y, Z (LSM6DSO)
} Sensor_Data;
```

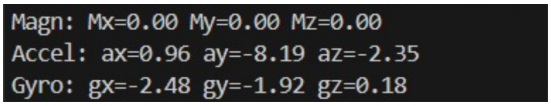
Description des membres (tous de type `double`) :
- `temperature` : Température ambiante en degrés Celsius (issue du HTS221).
- `humidity` : Humidité relative en pourcentage (issue du HTS221).
- `Mx, My, Mz` : Composantes du champ magnétique selon les axes X, Y, Z (issues du LIS2MDL, unité à préciser, ex : Gauss).
- `ax, ay, az` : Composantes de l'accélération linéaire selon les axes X, Y, Z (issues du LSM6DSO, unité : m/s^2).
- `gx, gy, gz` : Composantes de la vitesse angulaire selon les axes X, Y, Z (issues du LSM6DSO, unité : rad/s ou dps).

4.3.4 Validation Unitaire

La validation unitaire a permis de confirmer le bon fonctionnement de chaque capteur indépendamment :
- Vérification de l'initialisation correcte des capteurs au démarrage (absence de messages d'erreur des drivers, détection sur le bus I2C si nécessaire).
- Confirmation de la lecture de données cohérentes via les logs système (voir Figure 13). Les valeurs reçues ont été comparées aux attentes physiques :
  - **HTS221 :** Comparaison des valeurs de température et d'humidité lues avec les conditions ambiantes ou un thermomètre/hygromètre de référence.
  - **LSM6DSO (Accéléromètre) :** Vérification que l'axe Z avoisine 9.8 m/s^2 lorsque la carte est à plat et immobile, et que les valeurs varient logiquement lors de l'inclinaison ou de secousses.
  - **LSM6DSO (Gyroscope) :** Vérification que les valeurs sont proches de zéro à l'arrêt et réagissent correctement aux rotations.
  - **LIS2MDL (Magnétomètre) :** Vérification de la lecture de valeurs non nulles et de leur variation en présence de champs magnétiques ou lors de la rotation de la carte par rapport au champ magnétique terrestre.
- Test de la réactivité des capteurs aux changements (mouvements, rotations, souffle d'air chaud/humide pour HTS221).



(a) Logs HTS221 (Temp/Hum).



(b) Logs LSM6DSO (Accel/Gyro) et LIS2MDL (Magnétomètre)

FIGURE 13 – Exemples de logs système montrant les données des capteurs I2C lors de la validation unitaire.

FIGURE 14 – Concordance des données capteurs

4.4 Module Horloge Temps Réel (RTC - RV-8263-C8)

La gestion du temps via le RTC externe a été mise en place comme suit.

4.4.1 Configuration

Activation du driver `I2C` approprié et intégration d'un driver pour le `RV-8263-C8` (natif ou custom) dans `Zephyr`. Déclaration du composant RTC dans l'overlay DeviceTree sur le bus `I2C` adéquat.

4.4.2 Implémentation

Utilisation des fonctions du driver RTC pour lire et écrire l'heure/date. Mise en place d'une synchronisation éventuelle de l'heure système Zephyr au démarrage.

4.4.3 Fonctionnement du chronomètre

Le chronomètre est une fonctionnalité permettant d'afficher le temps qui s'écoule en heures, minutes et secondes. Dans notre cas, il est conçu pour fonctionner sur un système temps réel basé sur **Zephyr**, et utilise **LVGL** pour l'affichage graphique.

Son fonctionnement repose sur un compteur interne qui s'incrémente chaque seconde. À chaque incrément, le chronomètre calcule le temps écoulé en divisant ce compteur en heures, minutes et secondes, puis met à jour l'interface utilisateur en conséquence.

Techniquement, ce chronomètre est géré par le module `chrono.c`. Il utilise une tâche différée (un `k_work_delayable` dans Zephyr) qui s'exécute de manière répétée toutes les secondes. Cette tâche appelle une fonction qui :

— Incrémente le compteur de secondes,
— Calcule les heures, minutes et secondes correspondantes,
— Met à jour les étiquettes affichées à l'écran à l'aide des objets LVGL.

Lorsque l'on initialise le chronomètre via `chrono_init`, on lui fournit les objets d'interface (labels LVGL) qui afficheront les heures, minutes et secondes. Ensuite, on peut

démarrer le chronomètre avec la fonction `chrono_start`, qui lance la première exécution de la tâche. Cette tâche va ensuite se replanifier automatiquement chaque seconde, assurant une mise à jour continue.

On peut aussi l'arrêter avec `chrono_stop`, ou le réinitialiser avec `chrono_reset`, ce qui remet le compteur à zéro et réaffiche `"00:00:00"` à l'écran.

Ce fonctionnement permet de réaliser un chronomètre efficace, sans bloquer le reste du système, et avec une séparation claire entre la logique du temps et l'affichage graphique.

4.4.4 Validation

Les tests ont validé la communication `I2C` et la capacité à lire et écrire l'heure/date via des commandes et l'observation des logs (voir Figure 15). La persistance de l'heure après coupure d'alimentation du `nRF5340` a été vérifiée.

FIGURE 15 – Exemple de logs validant la lecture et l'écriture de l'heure RTC.

4.5 Interface Utilisateur Graphique (Écran TFT + LVGL + Squareline)

La mise en œuvre de l'interface utilisateur graphique (GUI) sur l'écran TFT couleur a constitué une étape majeure du développement, visant à fournir un retour visuel clair et une interaction intuitive avec l'utilisateur pour l'affichage des données capteurs et le contrôle des fonctionnalités. La bibliothèque `LVGL` (Light and Versatile Graphics Library) a été choisie pour sa légèreté et son adéquation aux systèmes embarqués, tandis que `Squareline Studio` a facilité la conception visuelle.

4.5.1 Configuration

La configuration matérielle et logicielle a nécessité plusieurs étapes :

- Activation des drivers de communication bas niveau via Kconfig : `SPI` pour l'écran et `I2C` ou `SPI` pour le contrôleur tactile (selon le matériel).
- Activation des drivers spécifiques aux composants : driver d'affichage (ex : `GC9307`, `ILI9341`, etc.) et driver tactile (ex : `STMPE610`, `FT6X06`, etc.).
- Configuration précise des broches (pins) utilisées par l'écran et le tactile dans l'overlay DeviceTree (`.overlay`) du projet pour correspondre au schéma matériel.
- Activation et configuration de la bibliothèque `LVGL` via Kconfig.
- Mise en place de l'intégration spécifique à Zephyr pour `LVGL`, notamment en liant les fonctions de bas niveau de Zephyr pour l'affichage (*display binding*) et la lecture des entrées tactiles (*input binding*) aux API de `LVGL`.

4.5.2 Structure des données Tactiles

Cette structure (définie de manière anonyme dans le code mais associée à la variable globale `touch_point`) est utilisée pour stocker l'état le plus récent de l'entrée tactile, mis à jour par le callback d'événements `touch_event_callback`.

Définition C : Données Tactiles

```c
/* Structure (anonyme) pour le point de contact */
static struct {
    size_t x;       // Coordonnée X
    size_t y;       // Coordonnée Y
    bool pressed;   // État (pressé / relâché)
} touch_point;
```

Description des membres :
— `x` : Coordonnée horizontale du point de contact (type `size_t`).
— `y` : Coordonnée verticale du point de contact (type `size_t`).
— `pressed` : Indicateur booléen de l'état tactile (`true` si l'écran est actuellement pressé, `false` sinon).

4.5.3 Implémentation

Le développement de l'interface s'est articulé autour des points suivants :
— **Initialisation :** Démarrage de la bibliothèque `LVGL` et initialisation des drivers d'affichage et tactile configurés précédemment.
— **Gestionnaire de Tâches LVGL :** Création et lancement d'une tâche Zephyr dédiée à l'appel périodique de `lv_timer_handler()`, fonction essentielle au fonctionnement de `LVGL` (gestion des animations, des événements, rafraîchissement).
— **Conception Visuelle :** Utilisation de l'outil `Squareline Studio` pour concevoir graphiquement les différents écrans de l'interface (widgets, agencement, styles). Exportation du code C généré par l'outil.
— **Intégration du Code UI :** Incorporation du code UI généré dans le projet Zephyr.
— **Liaison Données et Événements :** Connexion des widgets graphiques (labels, barres de progression, boutons...) aux variables et données pertinentes de l'application (valeurs capteurs, état BLE, etc.) pour un affichage dynamique. Implémentation des fonctions *callback* pour gérer les événements utilisateur (appui sur un bouton, geste tactile), reliant ainsi les interactions sur l'écran aux actions logiques de l'application.

4.5.4 Validation

La validation de l'interface graphique a permis de vérifier plusieurs aspects :
— Affichage correct et conforme des différents écrans tels que conçus dans `Squareline Studio` (voir Figure 16).
— Réactivité et précision de l'interface tactile lors des interactions utilisateur.
— Mise à jour dynamique et fluide des informations affichées (ex : valeurs capteurs en temps réel).
— Navigation fonctionnelle et logique entre les différents écrans de l'application.

FIGURE 16 – Captures d'écran des différents écrans de l'interface graphique validés.

4.6 Implémentation des Services et Caractéristiques GATT

L'implémentation GATT est au cœur de l'échange de données BLE. Elle a été réalisée en plusieurs étapes :

1. **Initialisation du Stack :** Démarrage du sous-système Bluetooth via la fonction `bt_enable`.
2. **Publicité (Advertising) :** Configuration et démarrage de la publicité BLE (`bt_le_adv_start`) pour rendre le dispositif découvrable, en annonçant son nom et les services principaux disponibles (notamment le service standard ESS et le service personnalisé).
3. **Service Standardisé (Environmental Sensing Service - ESS) :**
   — Définition du service ESS standard (UUID 16 bits : `0x181A`).
   — Ajout des caractéristiques standard pour la Température (UUID `0x2A6E`) et l'Humidité (UUID `0x2A6F`), avec les propriétés `READ` et `NOTIFY`.
   — **Formatage des Données Standard :** Une attention particulière a été portée au formatage des données pour ces caractéristiques, conformément aux spécifications du Bluetooth SIG. Par exemple, la température doit être envoyée comme un entier signé de 16 bits (`sint16`) représentant la valeur en degrés Celsius multipliée par 100 (résolution de 0.01°C). Une erreur initiale d'affichage dans l'application cliente (valeurs aberrantes comme 131.07°C) a été corrigée en implémentant cette mise à l'échelle et en s'assurant de l'ordre correct des octets (Little Endian) lors de l'envoi, notamment en utilisant `sys_put_le16` pour remplir le buffer de notification.
4. **Service Personnalisé (Ex : Motion Service) :**
   — Pour les données non couvertes par des caractéristiques standard (accéléromètre, gyroscope, magnétomètre), un service personnalisé a été créé.

— Définition d'UUIDs personnalisés 128 bits pour le service et ses caractéristiques (ex : `F0BA1000-...` pour le service, `F0BA1001-...` pour l'accéléromètre).
— **Formatage des Données Personnalisées :** Un format spécifique a été défini pour chaque caractéristique. Pour l'accéléromètre (LSM6DSO), les données X, Y, Z sont envoyées sous forme de 6 octets consécutifs, représentant trois entiers signés de 16 bits (`int16_t`) en Little Endian. Ces entiers correspondent aux valeurs en m/s^2 multipliées par 100 pour conserver une précision de deux décimales. Un format similaire a été adopté pour le gyroscope et le magnétomètre.

5. **Intégration des Données Capteurs Réelles :** Les données simulées initiales ont été remplacées par les lectures réelles des capteurs (HTS221, LSM6DSO, LIS2MDL) en utilisant l'API Sensor de Zephyr (`sensor_sample_fetch`, `sensor_channel_get`, `sensor_value_to_double`).
6. **Notifications :** L'envoi des données capteurs vers le client connecté est principalement réalisé via des notifications BLE (`bt_gatt_notify`), déclenchées périodiquement dans la boucle principale (ou potentiellement par des triggers capteurs pour une meilleure efficacité énergétique).
7. **Callbacks GATT et Connexion :** Implémentation des callbacks nécessaires pour gérer les lectures (`read_...`), les écritures sur les descripteurs CCC (Client Characteristic Configuration) pour l'activation/désactivation des notifications (`..._ccc_cfg_changed`), ainsi que les événements de connexion (`connected`) et déconnexion (`disconnected`).

4.7 Validation et Interprétation des Données

Les tests fonctionnels ont été menés à l'aide d'un smartphone Android exécutant l'application `nRF Connect for Mobile`, un outil standard pour le développement et le débogage BLE.

— La validation a confirmé le bon fonctionnement de la publicité, de la connexion/déconnexion, et de la structure des services et caractéristiques GATT.
— Les valeurs des caractéristiques standard (Température, Humidité) étaient affichées correctement après la correction du formatage des données.
— Pour les caractéristiques personnalisées (Accéléromètre, etc.), `nRF Connect` affiche initialement les données sous forme d'octets bruts hexadécimaux (par exemple, `(0x) C8-FF-0D-00-6C-00` pour l'accéléromètre), comme illustré en Figure 17.
— Il a été nécessaire de **configurer manuellement `nRF Connect`** pour chaque caractéristique personnalisée. En spécifiant le format attendu (par exemple, trois champs de type `int16le` pour l'accéléromètre), l'application a pu parser correctement les octets reçus et afficher les valeurs numériques correspondantes (ex : -56, 13, 108). Ces valeurs représentent les données mises à l'échelle (m/s^2 * 100), nécessitant une division par 100 pour retrouver l'unité physique originale.

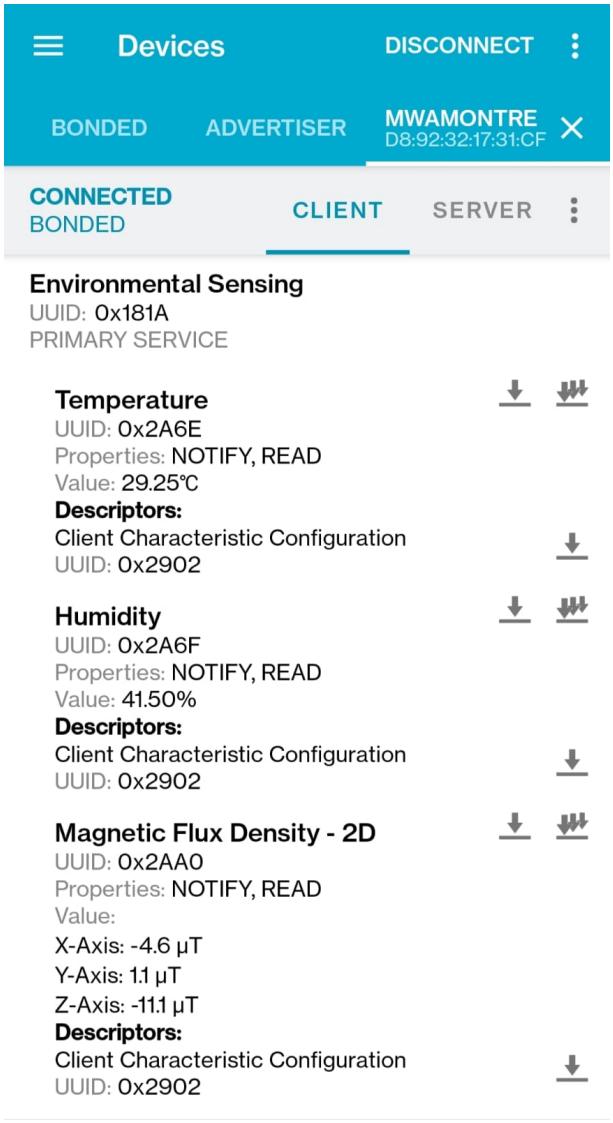


FIGURE 17 – Validation des caractéristiques personnalisées avec nRF Connect.

5 Composition de l'equipe

TABLE 4 – Composition et responsabilités dans l'équipe projet.

Membre	**Responsabilités Clés**
[Maxime MORET]	- Analyse préliminaire - Implémentation et Tests fonctionnels des capteurs I2C - Mise en place de la structure globale des threads - Implémentation et test de la RTC - Implémentation du touchscreen
[Koundeme Nobel DJESSOU]	- Analyse préliminaire - Rédaction du rapport - Implémentation et Tests fonctionnels du BLE - Tests unitaires
[Rayane DJENADOU]	- Analyse préliminaire - Implémentation et Test fonctionnels de l'interface graphique - Implémentation du chronomètre - Test Unitaires

6 Avis critique sur le projet

6.1 Points forts

Le projet présente une architecture logicielle complète et modulaire, intégrant plusieurs fonctionnalités avancées :

— Affichage graphique fluide et multi-écran via LVGL.
— Acquisition et traitement de plusieurs capteurs en parallèle (accéléromètre, magnétomètre, capteur d'humidité/température).
— Synchronisation des données avec l'affichage en temps réel et transmission Bluetooth.
— Intégration de la RTC avec affichage de l'heure/date.
— Séparation claire des tâches à l'aide de threads indépendants (UI, capteurs, BLE, RTC).

Le système est déjà fonctionnel et suffisamment robuste pour démontrer un fonctionnement cohérent sur le matériel cible.

6.2 Points à améliorer

Malgré la qualité de l'implémentation, certaines limitations subsistent :

— La logique de mise en veille automatique reste incomplète et ne fonctionne pas.
— Le tactile pourrait être davantage filtré pour éviter les interactions non désirées.
— Le système pourrait tirer parti d'un gestionnaire d'énergie plus centralisé (machine à états ou module dédié) et on n'aurait pu faire des tests pour voir la consommation.

— Des protections mémoire et synchronisations plus strictes sont nécessaires pour garantir la stabilité à long terme(sémaphore et mutex).

6.3 Perspectives d'évolution

Le projet possède une base solide, apte à être étendue vers :
— Une gestion d'énergie complète avec états basse consommation.
— Une interface BLE plus riche (ajout de commandes ou configuration à distance).
— Un menu interactif pour le changement de vue, couplé à des animations LVGL.
— Un système de journalisation ou d'enregistrement local (flash ou carte SD).

Conclusion

Au cours de ce projet, nous avons conçu une montre connectée fonctionnant sous *Zephyr RTOS*, intégrant plusieurs capteurs (HTS221, LSM6DSO, LIS2MDL), une interface graphique tactile avec *LVGL*, et un module de communication *Bluetooth Low Energy (BLE)*. Nous avons mis en place une architecture logicielle multi-threads, permettant l'acquisition, le traitement et l'affichage en temps réel des données environnementales et de mouvement. Ces données sont également transmises à une application mobile via un service BLE personnalisé, testé avec succès sur *nRF Connect* et *nRF Toolbox*.

L'interface utilisateur, entièrement développée avec *LVGL*, permet une navigation tactile fluide entre différentes vues. Ce travail nous a permis de mettre en œuvre une communication fluide entre capteurs, interface graphique et connectivité sans fil, en respectant les contraintes d'un système embarqué.

Nous avons tenté de mettre en place une mise en veille automatique après une période d'inactivité, en utilisant un timer matériel (`NRFX_TIMER`). Malgré une configuration conforme à la documentation, le timer n'a pas fonctionné comme attendu, et la fonctionnalité de mise en sommeil n'a pas pu être finalisée. Ce point reste une piste d'amélioration importante pour optimiser l'autonomie du système.

Ce projet nous a permis de consolider nos compétences en développement embarqué, en gestion de capteurs, en conception d'interface graphique, et en communication BLE. Il constitue une base solide pour de futures extensions, notamment en matière de gestion énergétique, de traitement de données locales, ou d'intégration avec des services cloud.

Le code du projet peut être retrouvé ici : https://github.com/M0x3m/Montre_Connectee_Projet